图文 047、高级工程师的硬核技能：JVM的Full GC日志应该怎么看？

1433 人次阅读 2019-08-16 07:00:00

详情 评论

高级工程师的硬核技能：

JVM的Full GC日志应该怎么看？

给大家推荐一套质量极高的Java面试训练营课程：

作者是中华石杉，石杉老哥是我之前所在团队的 Leader ，骨灰级的技术神牛！

大家可以点击下方链接，了解更多详情，并进行试听：

21天互联网Java进阶面试训练营（分布式篇）

1、前文回顾

之前两篇文章已经给大家分析清楚了对象是如何进入老年代的，接着我们就给大家演示一下，老年代的GC是如何触发的。

2、示例代码

```java
public class Demo1 {

    public static void main(String[] args) {
        byte[] array1 = new byte[4 * 1024 * 1024];
        array1 = null;

        byte[] array2 = new byte[2 * 1024 * 1024];
        byte[] array3 = new byte[2 * 1024 * 1024];
        byte[] array4 = new byte[2 * 1024 * 1024];
        byte[] array5 = new byte[128 * 1024];

        byte[] array6 = new byte[2 * 1024 * 1024];
    }

}
```

3、GC日志

我们需要采用如下参数来运行上述程序：

“-XX:NewSize=10485760 -XX:MaxNewSize=10485760 -XX:InitialHeapSize=20971520 -XX:MaxHeapSize=20971520 -XX:SurvivorRatio=8 -XX:MaxTenuringThreshold=15 -XX:PretenureSizeThreshold=3145728 -XX:+UseParNewGC -XX:+UseConcMarkSweepGC -XX:+PrintGCDetails -XX:+PrintGCTimeStamps -Xloggc:gc.log”

这里最关键一个参数，就是“-XX:PretenureSizeThreshold=3145728”

这个参数要设置大对象阈值为3MB，也就是超过3MB，就直接进入老年代。

运行之后会得到如下GC日志：

“0.308: [GC (Allocation Failure) 0.308: [ParNew (promotion failed): 7260K->7970K(9216K), 0.0048975 secs]0.314: [CMS: 8194K->6836K(10240K), 0.0049920 secs] 11356K->6836K(19456K), [Metaspace: 2776K->2776K(1056768K)], 0.0106074 secs] [Times: user=0.00 sys=0.00, real=0.01 secs]

Heap

par new generation total 9216K, used 2130K [0x00000000fec00000, 0x00000000ff600000, 0x00000000ff600000)

eden space 8192K, 26% used [0x00000000fec00000, 0x00000000fee14930, 0x00000000ff400000)

from space 1024K, 0% used [0x00000000ff500000, 0x00000000ff500000, 0x00000000ff600000)

to space 1024K, 0% used [0x00000000ff400000, 0x00000000ff400000, 0x00000000ff500000)

concurrent mark-sweep generation total 10240K, used 6836K [0x00000000ff600000, 0x0000000100000000, 0x0000000100000000)

Metaspace used 2782K, capacity 4486K, committed 4864K, reserved 1056768K

class space used 300K, capacity 386K, committed 512K, reserved 1048576K”

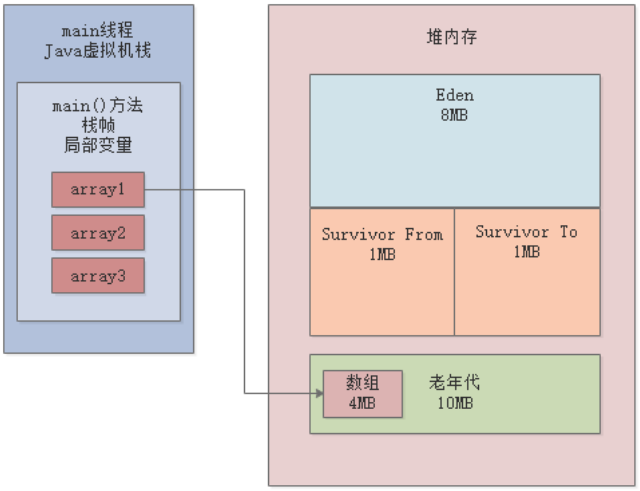
4、一步一图分析日志

首先我们看如下代码：

```
byte[] array1 = new byte[4 * 1024 * 1024];
array1 = null;
```

这行代码直接分配了一个4MB的大对象，此时这个对象会直接进入老年代，接着array1不再引用这个对象

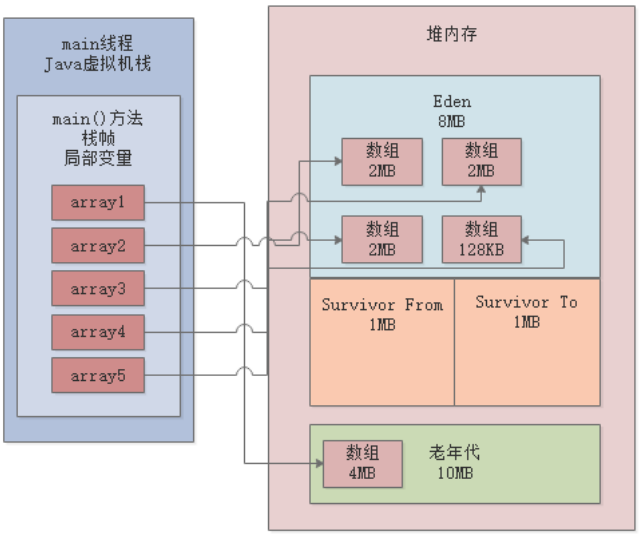
此时如下图所示。



接着看下面的代码：

```
byte[] array2 = new byte[2 * 1024 * 1024];
byte[] array3 = new byte[2 * 1024 * 1024];
byte[] array4 = new byte[2 * 1024 * 1024];
byte[] array5 = new byte[128 * 1024];
```

连续分配了4个数组，其中3个是2MB的数组，1个是128KB的数组，如下图所示，全部会进入Eden区域中。



接着会执行如下代码：byte[] array6 = new byte[2 * 1024 * 1024];。此时还能放得下2MB的对象吗？不可能了，因为Eden区已经放不下了。因此此时会直接触发一次Young GC。

我们看下面的GC日志：ParNew (promotion failed): 7260K->7970K(9216K), 0.0048975 secs

这行日志显示了，Eden区原来是有7000多KB的对象，但是回收之后发现一个都回收不掉，因为上述几个数组都被变量引用了。

所以此时大家都知道，一定会直接把这些对象放入到老年代里去，但是此时老年代里已经有一个4MB的数组了，还能放的下3个2MB的数组和1个128KB的数组吗？

明显是不行的，此时一定会超过老年代的10MB大小。

所以此时我们看gc日志：

[CMS: 8194K->6836K(10240K), 0.0049920 secs] 11356K->6836K(19456K), [Metaspace: 2776K->2776K(1056768K)], 0.0106074 secs]

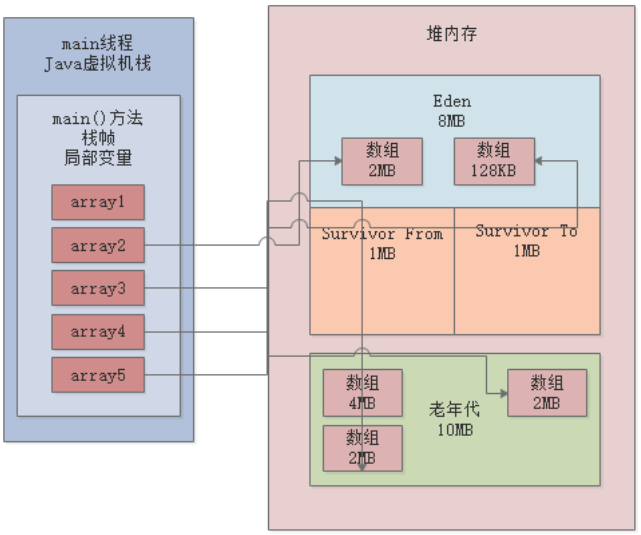
大家可以清晰看到，此时执行了CMS垃圾回收器的Full GC，我们之前讲过Full GC其实就是会对老年代进行Old GC，同时一般会跟一次Young GC关联，还会触发一次元数据区（永久代）的GC。

在CMS Full GC之前，就已经触发过Young GC了，此时大家可以看到此时Young GC就已经有了，接着就是执行针对老年代的Old GC，也就是如下日志：

CMS: 8194K->6836K(10240K), 0.0049920 secs

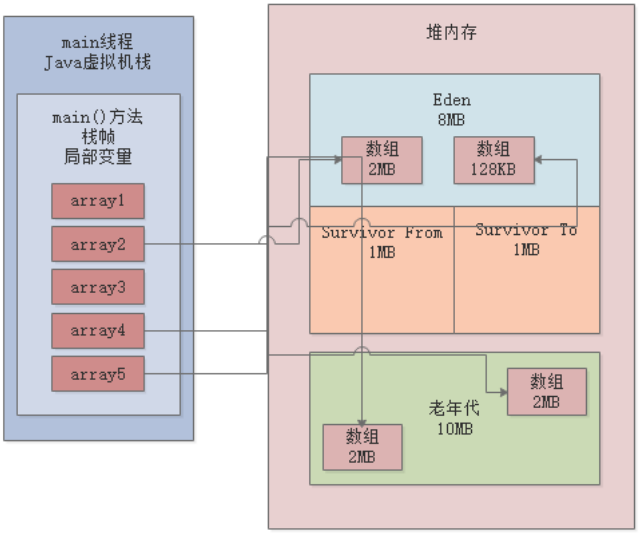
这里看到老年代从8MB左右的对象占用，变成了6MB左右的对象占用，这是怎么个过程呢?

很简单，一定是在Young GC之后，先把2个2MB的数组放入了老年代，如下图。

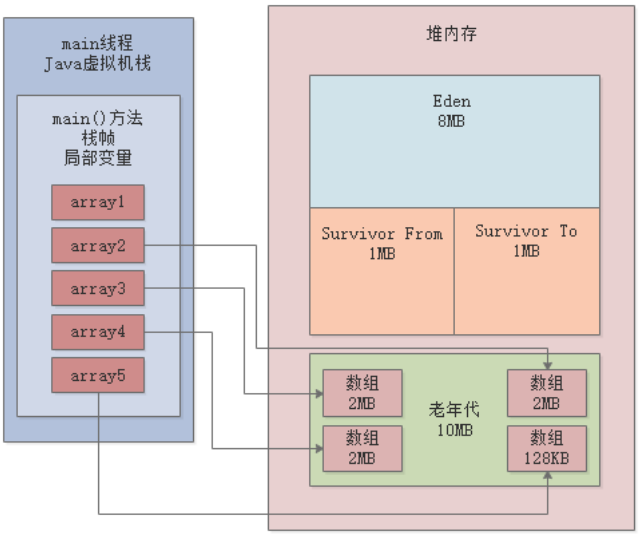


此时要继续放1个2MB的数组和1个128KB的数组到老年代，一定会放不下，所以此时就会触发CMS的Full GC

然后此时就会回收掉其中的一个4MB的数组，因为他已经没人引用了，如下图所示。

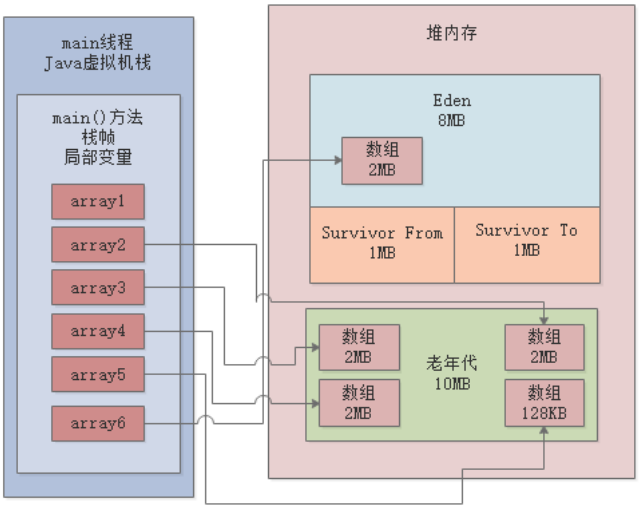


接着放入进去1个2MB的数组和1个128KB的数组，如下图所示。



所以大家再看CMS的垃圾回收日志：CMS: 8194K->6836K(10240K), 0.0049920 secs，他是从回收前的8MB变成了6MB，就是上图所示。

最后在CMS Full GC执行完毕之后，其实年轻代的对象都进入了老年代，此时最后一行代码要在年轻代分配2MB的数组就可以成功了，如下图。



5、本文总结

本文给大家又讲解了一个触发老年代GC的案例，就是年轻代存活的对象太多放不下老年代了，此时就会触发CMS的Full GC，大家可以清晰的看到全过程。

6、今日思考题

今天留给大家的思考题，就是让大家可以自己写代码模拟出来另外几种老年代GC的场景

其中一个就是在触发Young GC之前，可能老年代可用空间小于了历次Young GC后升入老年代的对象的平均大小，就会在Young GC之前，提前触发Full GC。

还有一个，就是老年代被使用率达到了92%的阈值，也会触发Full GC。

其实说实话，很多场景并不太容易用代码模拟出来，但是大家还是尽量尝试一下，因为这样可以增强大家对这里原理的一个理解。

End